Knowl Inf Syst (2011) 28:665–685
DOI 10.1007/s10115-010-0360-6

REGULAR PAPER

SimClus: an effective algorithm for clustering with a lower bound on similarity

Mohammad Al Hasan · Saeed Salem · Mohammed J. Zaki

Received: 17 August 2009 / Revised: 18 March 2010 / Accepted: 25 October 2010 /
Published online: 16 November 2010

Abstract Clustering algorithms generally accept a parameter k from the user, which determines the number of clusters sought. However, in many application domains, like document categorization, social network clustering, and frequent pattern summarization, the proper value of k is difficult to guess. An alternative clustering formulation that does not require k is to impose a lower bound on the *similarity* between an object and its corresponding cluster representative. Such a formulation chooses exactly one representative for every cluster and minimizes the representative count. It has many additional benefits. For instance, it supports overlapping clusters in a natural way. Moreover, for every cluster, it selects a representative object, which can be effectively used in summarization or semi-supervised classification task. In this work, we propose an algorithm, **SimClus**, for clustering with lower bound on similarity. It achieves a $O(\log n)$ approximation bound on the number of clusters, whereas for the best previous algorithm the bound can be as poor as $O(n)$. Experiments on real and synthetic data sets show that our algorithm produces more than 40% fewer representative objects, yet offers the same or better clustering quality. We also propose a dynamic variant of the algorithm, which can be effectively used in an on-line setting.

Keywords Dominating set · Overlapping clustering · Set cover · Star clustering

Part of this work was done when the first author was at eBay Research Labs, San Jose, CA.

M. A. Hasan (✉)
Indiana University–Purdue University, Indianapolis, IN, USA
e-mail: alhasan@cs.iupui.edu

S. Salem
Department of Computer Science, North Dakota State University, Fargo, ND, USA
e-mail: saeed.salem@ndsu.edu

M. J. Zaki
Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY, USA
e-mail: zaki@cs.rpi.edu

1 Introduction

In many application domains that involve clustering, it is difficult to obtain a good guess for the number of clusters. Consider a scenario where the top search results obtained through a query on a search engine are clustered to display the result in a hierarchical manner [29]. The documents at the top level are the representative documents for the clusters that contain the documents lower in the hierarchy. However, the number of clusters in this clustering task cannot be fixed before-hand, rather they should be chosen based on homonymy to the query word. The number may also depend on the size and the dissonance of the documents in the result-set. Data compression, summarization or representative sampling poses another compelling application domains. For example, in frequent pattern summarization [9,28], depending on the data set and the support criteria, the number of frequent patterns changes dramatically, so a compression (summarization) algorithm that just returns the top-k representative patterns through clustering is not that intuitive, since the user has no idea regarding a value of k that would return a potentially useful summary set. Besides these, in many dynamic platforms, like newsgroups and blogosphere, where the number of topics is typically unknown, finding a right value of k is always challenging. Furthermore, it is also not feasible to try out different values of k and analyze the result-set (off-line) due to the highly dynamic nature of these domains.

Even if the value of k is known (for instance, in a static document collection where the number of topics is fixed), it still may not be good to partition the data directly using k partitions. Recent studies on document categorization suggest that documents are usually sampled from a nonlinear low-dimensional manifold that is embedded in the high-dimensional ambient space [13]. In such cases, the cluster boundaries are highly irregular, so only one prototype (as in k-Means) may not be able to capture the arbitrary-shaped clusters. In such cases, multiple (the exact number depends on how irregular the cluster shape is) centers may be required to represent each cluster effectively. Actually the idea of finding smaller clusters and merging them together to obtain a desired number of final clusters has been proposed in the domain of arbitrary-shape clustering [10,18], community finding [25], and gene functional classification [16]. An alternative to the parameter k that defines the number of clusters, is to provide a lower bound, β, that defines the desired (minimum) similarity between a cluster member and a representative object in that cluster. Generally, the similarity measures that are used in clustering (Cosine similarity, Jaccard coefficient) have a typical range from 0 (for completely disjoint) to 1 (for identically similar), so a lower bound can be chosen sensibly. Also, a small random sample of objects can be used to find a rough estimate of the typical similarity in a collection of objects to enable a good guess of the lower bound value. The above clustering paradigm has many benefits. First, the lower bound of similarity between a member object and a representative object automatically imposes a similarity bound among the members of a cluster, which cannot be claimed for the typical EM-based local optimal clustering approaches (such as, k-Means). Second, for every cluster this paradigm provides a representative object. This fact is worthwhile, since clustering algorithms, as we have discussed earlier, are sometimes used to obtain representative objects from a collection of objects. Third, it supports overlapping clusters in a very natural way. Overlapping clustering is appealing in many domains [17]. For example, in information retrieval, documents are generally labeled as multi-topic; in social networks [23], a person may belong to multiple communities; in gene classification, one gene can be associated with many different functionalities, and so on. Fourthly, this clustering paradigm can easily be adapted to work in a dynamic setting, where new objects are added in or existing objects are removed from present clusters; which is, again, a very desirable feature in highly dynamic environments, like newsgroups and blogs. Finally, it supports semi-supervised classification; one needs to provide the labels of the representative objects in the supervision step from which the labels of the rest are automatically inferred. However, the number of representatives should be as small as possible, since the supervision cost increases with the number of labeled examples. Existing clustering approaches can only achieve a strict subset of the above benefits. For instance, the popular k-Means algorithm can find representatives, but it requires the k value as an input. Hierarchical clustering (HC) does not return representatives, but it works without a k value. In fact, HC generates a hierarchical tree, which denotes clustering at different levels of the hierarchy, so the choice of k can be made afterward. However, none of these algorithms supports overlapping clustering, neither do they work for dynamic clustering in their standard setting. In terms of time complexity, k-Means is linear (with respect to the number of objects) only for vector-based data. For non-vector data, like sets and graphs, the k-Medoids variant needs to be employed, which has a quadratic time complexity [24]. Complexity of HC is also at least quadratic. The lower bound similarity clustering proposed in this research computes the entire similarity matrix; thus, in the worst case it requires quadratic time. However, the complexity of the similarity computation can be substantially reduced by adopting recently proposed techniques [8].

1.1 Contribution

We prove that the optimization problem of clustering with lower bound on similarity is NP-Hard and propose a greedy solution, named **SimClus**, that achieves a $O(\log n)$ approximation bound (n is the size of the object-set). We also show that this bound for the existing best algorithm (*star* clustering) is as worse as $O(n)$. We then propose a variant of **SimClus** that is suitable for clustering in a dynamic setting. We experiment with different synthetic and real-world data sets; like, random graphs, newsgroup document-set, and e-commerce query data set. In all experiments, **SimClus** achieves similar or better clustering performance in comparison with *star* clustering algorithm with more than 40% fewer representatives. It also outperforms traditional clustering algorithms (k-Medoids, HC) in terms of cluster quality and execution time.

2 Background

Consider, a set of objects, $\mathcal{O}$ and a similarity function, $sim : \mathcal{O} \times \mathcal{O} \rightarrow [0, 1]$, such that for any $x \in \mathcal{O} : sim(x, x) = 1$. The objective is to cluster the objects in $\mathcal{O}$ such that the objects in a cluster are at least β-similar for a user defined $\beta \in [0, 1]$ with minimum number of clusters.

The above formulation leads to an interesting graph problem. To see this, consider, $G(V, E)$ to be a graph whose vertices are the objects to be clustered. An edge $e(u, v) \in E$ implies that the similarity between vertex u and v is at least β i.e., $sim(u, v) \geq \beta$. In further discussion, we will call this graph the *β-similarity graph* or *similarity graph* when the value of β is not important in the context of the discussion. Now, any clique in this graph can be taken as one cluster in some clustering, since the distances between the elements in this clique satisfy the required pair-wise similarity constraints. The clustering objective then becomes to cover the entire graph G by a minimum number of cliques. Here, *covering* stands for the fact that the union of vertices belonging to these cliques is equal to the vertex-set V of the graph G. However, this formulation of clustering is difficult to solve; in fact, it leads to an

NP-Complete problem, named covering a graph by cliques, which cannot be approximated in polynomial time [30].

A relaxation of the above problem can be obtained which requires the similarity bound (β) to hold only between the cluster elements and a fixed center object belonging to that cluster. The center object is the representative for the corresponding cluster. Thus if a cluster has m elements, out of all $\binom{m}{2}$ similarities, $m - 1$ are guaranteed to be greater than or equal to β. Nevertheless, all the m objects in the cluster are sufficiently similar due to triangular inequality or other form of transitive bounds. The diameter of a cluster in the similarity graph is at most 2. We call it *lower bound similarity clustering* (*LBSC*), which is the main focus of this paper.

LBSC seeks exactly one cluster center (representative object) for every cluster. Thus, a center object c together with all the objects s such that $sim(c, s) \geq \beta$, form a cluster. The set of objects, s that satisfy the above inequality are called β-similar objects with respect to the object c. If s is β-similar to multiple centers, it belongs to multiple clusters. Thus, this model naturally supports overlapping clusters. Since every object s belongs to at least one cluster, s is β-similar to at least one representative object, say c. In that case, we say that *c covers s*. In the worst case, s can cover itself by being its own representative. Thus, the clustering objective is to cover all the objects with the smallest number of center objects. A center always covers itself.

Unfortunately, the optimization task of LBSC is also NP-Hard. To prove this, we can reduce the vertex dominating set (a known NP-Complete problem) to the decision version of this problem. A *vertex dominating set* in a graph is defined as a set of vertices such that every vertex of the graph is either a member of this set or is adjacent to a member of this set. Now the following lemma holds.

Lemma 1 *For a given collection of objects $\mathcal{O}$, and a user-defined similarity threshold β, to determine whether there exists a set of representative objects, $\mathcal{C} \in \mathcal{O}$, of size k is NP-Complete.*

Proof We can reduce the vertex dominating set problem (known NP-Complete problem [15]) to this problem. For a given graph $G(V, E)$, assume that each vertex $v \in V$ represents an object in $\mathcal{O}$. For every edge $(v_1, v_2) \in E$, we consider that the similarity between the corresponding objects is at least β, i. e., v_1 is a cover for v_2 and vice-versa. Now, there exists a dominating set, R of size k or less in G, if and only if, we have a center set, $\mathcal{C}$, of size k or less such that all the non-center objects are covered by at least one center. It is so, because for any $v \notin R$, there exists $u \in R$ adjacent to it, Similarly, since every object $q \in \mathcal{O} \backslash \mathcal{C}$ belongs to some cluster, there exists a cluster center $p \in \mathcal{C}$ such that $sim(p, q) \geq \beta$. Thus, every object in $\mathcal{O}$ is either in the center set or is covered by at least one cluster center. Hence, the Lemma is proved. □

The above relation to the dominating set problem suggests a graph-based formulation. From $\binom{n}{2}$ similarity values for a set of n objects, we can first construct a β-similarity graph. Then, in this graph, we need to find a vertex dominating set, which would constitute the representative set of the desired clustering. Figure 1a shows a β-similarity graph of a set of objects for some β. Figure 1b shows a LBSC clustering of these objects with 3 clusters. Each cluster is marked with a dotted closed curve and the representative objects are marked with gray color. Also note that the clusters are overlapping as objects 3, 4, and 5 belong to multiple clusters.

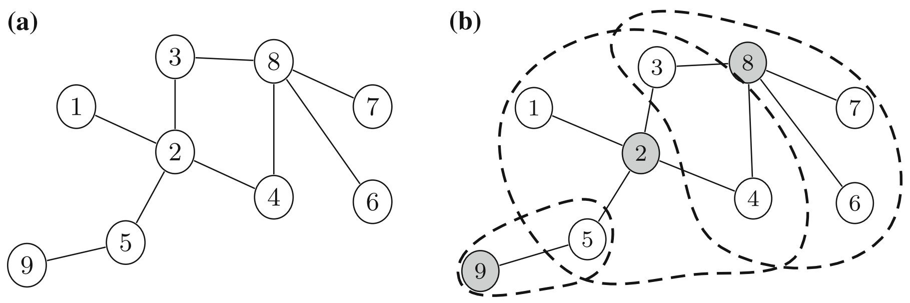

Fig. 1 An example of overlapping clustering with lower bound similarity. **a** A β-similar graphical representation of a set of objects. **b** A minimum distance threshold clustering

In further discussion, objects may be referred to as vertex or node in a β-similarity graph in relation to this graph-based formulation. Similarly, the member of cluster representatives may also be referred to as dominating set, center-nodes, or cluster-centers.

Star clustering: Aslam et al. [2] solved the above clustering formulation by a greedy algorithm, named *star clustering*. The greedy algorithm works as follows. It sorts the vertices in descending order of their degrees; it then selects the first vertex in the sorted order as one of the cluster centers. Any other vertex covered by this one is deleted from the sorted list, and the procedure is repeated until all the vertices are covered. If the similarity matrix is provided and cost associated with its loading and storing is ignored, the runtime complexity of this algorithm is $O(|V|.\lg|V|)$ to perform the sorting. The clusters in Fig. 1(b) are obtained using the *star clustering* algorithm.

Limitations of star algorithm: The greedy solution that the *star* algorithm provides is obviously not optimal. In fact, the approximation bound can be very bad. To show that, we first prove the following Lemma regarding the solution obtained by the *star* algorithm.

Lemma 2 *The greedy star clustering algorithm always generates an independent dominating set.*

Proof To observe this, note that once it selects an object as a center-node, it considers all its adjacent vertices to be covered and removes those from the sorted list. Hence, in later iterations they never appear in the dominating set. So, the result-set returned by the algorithm is an independent set.[1] □

Note that, the independent set restriction is the main reason for the greedy algorithm to obtain a poor clustering with unnecessarily many star-centers. In Fig. 2, we show an example that illustrates the poor bound of the star clustering algorithm. Instead of choosing {2, 8} as center-nodes, star algorithm chooses {1, 3, 4, 5, 8}, to respect the independence set condition. This example can be generalized to obtain a bound in an asymptotic sense. Assume, we have n vertices arranged in two star-shapes (as in Fig. 2, each with $\frac{n}{2} - 1$ satellites and one star-center, where the star-centers in two star-shapes are adjacent to each other. In this case, the *star* algorithm would require $\frac{n}{2}$ centers, whereas the optimal LBSC would have only 2 centers; thus, the approximation ratio can be as bad as $O(n)$.

The independent set formulation has several further consequences on the quality of representative set obtained by the *star* algorithm. In fact, the objects that it chooses as cluster centers are, in many cases, not really the true representative objects. For instance in Fig. 2,

[1] An *independent set* is a set of vertices in a graph such that no pair of vertices in this set are adjacent.

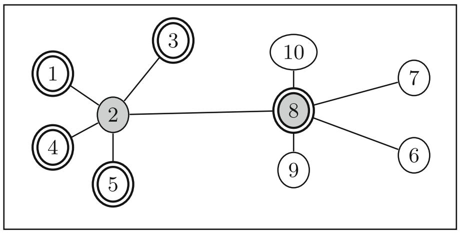

Fig. 2 Poor approximation bound of *star* clustering algorithm, it chooses 5 objects (double-lined) as cluster-centers, but the optimal cluster-center has size 2 (*gray color*)

the object 2 and 8 are the best representative objects, but the *star* algorithm chooses 5 representative objects, out of which 4 represent (overlapping) clusters with only two elements.

3 The SimClus formulation

In this research, we propose a set cover formulation, named **SimClus** for LBSC. It is also greedy in nature, but with a better ($O(\log n)$) bound. So, generally it produces fewer cluster centers. Moreover, it produces better representative objects for each cluster.

First, we present the following Lemma that proves the connection of lower bound similarity clustering with the set cover problem [27].

Lemma 3 *LBSC can be reduced to a set cover problem, where the number of cluster-centers is equal to the size of the selected set.*

Proof Let us assume that $G(V, E)$ is the similarity graph of LBSC, where every vertex represents one object in $\mathcal{O}$. So, $|V| = |\mathcal{O}|$ and there is a one-to-one correspondence ($f : V \rightarrow \mathcal{O}$) between the vertex-set and the object-set. Now we reduce the LBSC problem to an instance of set-cover problem in polynomial time as below:

For each vertex $u \in V$, we construct an object set $s_u = f(u) \cup \{f(v) : v \in adj(u)\}$, where $adj(u)$ is the set of vertices adjacent to u in G. Let $\mathcal{S} = \{s_u : u \in V\}$. Obviously, $|\mathcal{S}| = |V|$ and there exists a one-to-one correspondence between the elements in $\mathcal{S}$ and the vertices in V. Now, an optimal solution to the set cover problem that chooses the minimum number of elements from $\mathcal{S}$ to cover all objects in $\mathcal{O}$ can be used to obtain an optimal solution to LBSC as follows. Assume $\mathcal{T} \subseteq \mathcal{S}$ is the optimal set cover and $\mathcal{C}$ is the cluster-centers of LBSC, which is initially empty. Now, for each $s_x \in \mathcal{T}$, we insert the corresponding vertex $x \in V$ in $\mathcal{C}$. Now, the minimality of $|\mathcal{T}|$ ensures the minimality of $|\mathcal{C}|$. Thus, $\mathcal{C}$ is an optimal solution for the LBSC problem and also, $|\mathcal{C}| = |\mathcal{T}|$. □

The set-cover problem is NP-Complete, but it can be solved using an efficient greedy algorithm that achieves a $O(\log n)$ approximation bound [27]. The greedy heuristic iteratively chooses the set that covers the largest number of uncovered elements until all the elements are covered. We use the above greedy algorithm to solve LBSC after reducing it to the set cover as described in the above proof. Thus, our algorithm solves LBSC with a $O(\log n)$ approximation bound that yields much fewer representatives. In the experimental section, we validate this claim using synthetic and real-life data sets.

For the dominating set problem, the hardness of approximation result is also available, which proves that unless $P = NP$, no polynomial time algorithm can have better than $O(\log n)$ approximation for this problem [22]. So, it is highly unlikely that a better bound over $\log n$ can be achieved for LBSC, using a polynomial algorithm.

```
SimClusClustering($G, n$):
1.  $covered(1:n) = 0$, $\mathcal{C} = \emptyset$
2.  $uncov\_cover\_set(o) = \{o \cup adj(o)\}, \forall o = 1 \ldots n$
3.  while $\exists k$ such that $covered(k) == 0$
4.      Find the set $\mathcal{P}$ with object $i$, where $i \notin \mathcal{C}$
            and $size(uncov\_cover\_set(i))$ is the largest
5.      Find the set $\mathcal{Q} \subseteq \mathcal{P}$, where $degree(i)$ is the largest
            for $i \in \mathcal{Q}$
6.      select $s$ arbitrarily form $\mathcal{Q}$
7.      $covered(s) = covered(s) + 1$
8.      $\mathcal{C} = \mathcal{C} \cup s$
9.      for each $y \in adj(s)$
10.         $covered(y) = covered(y) + 1$
11.     update $uncov\_cover\_set(o), \forall o \in adj(x)$,
                     where $x \in uncov\_cover\_set(s)$)
12. return $\mathcal{C}$
```

Fig. 3 Pseudo-code for SimClus: clustering with lower bound on similarity using greedy set cover

The main difference between star clustering and **SimClus** is that instead of choosing the uncovered object with the highest degree, the latter chooses the object that can cover the most uncovered elements. This drastically reduces the number of clusters. Thus, the clusters are more dense and hence, more informative.

Implementation details: The algorithm implementation follows the reduction that is proved in Lemma 3. At the beginning, all the objects are not covered and the center-set is empty. We then map each vertex, u, of LBSC similarity graph to a set s_u that contains itself and all objects that it can cover. To facilitate the greedy heuristic, the set $s_u, \forall u \in V$ contains only those objects that are uncovered. So, we name the set s_u as *uncovered cover-set* of u, which intuitively means that it holds those objects that are uncovered and can be covered by choosing u as a center. Hence, once an object is chosen as a center, the *uncovered cover-set* of all the non-center objects are updated by removing any object that is covered by the newly chosen center. In every iteration, a new center is selected using the above greedy criterion and the process is repeated until all the objects are covered. If there is a tie in the above criterion, we break the tie by selecting the object that has the largest degree in the similarity graph. If there is still a tie, we break it arbitrarily.

The pseudo-code for SimClus is provided in Fig. 3. It accepts a β-similarity graph G and returns the cluster-centers $\mathcal{C}$. The $covered$ vector counts the coverage of each object. The coverage of an object is the number of centers to which it is adjacent. A correct clustering requires that every existing object has the coverage strictly greater than zero. At initialization, the $covered$ vector is 0 and thus each object is noted as uncovered and the cluster-center set $\mathcal{C}$ is empty (line 1). The *uncovered cover-set* of every vertex in initialized in line 2. As more objects are chosen as cluster-centers, these sets are updated in line 11. The main loop (line 3) of the algorithm selects one object to be a new center according to the greedy criterion and terminates when all the objects are covered.

Example: Consider the similarity graph in Fig. 2. The *uncovered cover-set* of vertex 1 is {1, 2}; for vertex 2, it is {1, 2, 3, 4, 5, 8}, and for vertex 8 it is {2, 6, 7, 8, 9, 10}. At the beginning, the *uncovered cover-set* of both the vertices 2 and 8 are the highest with the same size. They also have equal degrees. So, we arbitrarily choose, say vertex 2, as a center. After updating the *uncovered cover-set* for all the objects, we have empty *uncovered cover-set* for 1,3,4 and 5 and only one object for 6,7,9 and 10. But, for object 8, *uncovered cover-set* contains {6, 7, 9, 10}. So, 8 is selected as the next center. At this point, all the objects are covered, and the algorithm terminates by returning {2, 8} as representatives.

Complexity analysis: SimClus computes the entire similarity matrix to find the β-similarity graph. So, the overall complexity of the algorithm is at least $O(|V|^2)$. If we consider that the similarity matrix is given, then the overall worst-case complexity can be computed in terms of the number of the centers obtained. Assume that the final clustering has k centers. Then the outer while loop (line 3) runs for k times. In each iteration, the best center can be obtained in $O(\lg |V|)$ time by using a max_heap data structure. The cover-set update affects only those vertices that are adjacent to the newly covered objects. For the sake of worse case analysis, if we assume that to be $O(|V|)$, the worst case complexity of the algorithm is $O(k.(\lg |V| + |V|)) = O(k|V|)$.

4 Dynamic SimClus algorithm

The static algorithm that is provided in the previous section requires that the entire β-similarity graph is available before the algorithm is applied. However, in many practical application scenarios, this requirement does not hold. For information retrieval, new documents can be added or old documents may be deleted from the repositories and the clustering may need to be updated. One option is to obviously re-cluster the objects by running the static clustering algorithm for every change in the collection. But, for most of the cases, the changes are only local; so re-computing the entire clustering is wasteful. Also note that re-computation dramatically changes the cluster-centers; so, if the objective of LBSC is to find representative objects in a dynamic setting, one may prefer the dynamic algorithm over the static algorithm, since the former retains a large portion of the representative set. It is also useful in adopting a lazy update scheme where a static re-clustering is applied intermittently after a fixed number of dynamic updates.

We like to clarify one subtle point regarding the dynamic model in the above paragraph. This model is not the same as the on-line model that is studied in theoretical computer science. In the on-line model, the decision that is made based on present information cannot be altered later when further information is available in the future. So in such setting, if a node is made representative, its status cannot be changed later. Nevertheless, an on-line version of LBSC is very difficult; in fact, the *competitive ratio* of it is as worse as $O(n-1)$ [19]. The dynamic model that we propose makes local changes on the status of a node (whether it is a representative or not). These changes are much cheaper compared to a total re-clustering which make the dynamic version more appealing for cluster updates. But, they are based on heuristics, so we do not expect a dynamic SimClus algorithm to satisfy the $O(\lg n)$ approximate guaranty that exists for the static SimClus (through the set cover formulation) regarding the number of clusters (representative objects). So, it may be worthwhile to perform a static update after a batch of dynamic updates to re-optimize the clustering.

In a dynamic environment, we assume that we have a solution for a lower bound similarity clustering. New requests for insertion or deletion of an object come in an arbitrary manner. We need to satisfy the requests efficiently while maintaining the minimality of representative set as much as possible. We are allowed to change the status of an object in either direction (a center can be made non-center and vice-versa).

Given the above scenario, the addition of new objects cannot be made based on the greedy criteria of the static algorithm presented in Fig. 3, as only one object (the new object) is uncovered and all vertices adjacent to that object have exactly the same size (one) for the *uncovered cover-set*. So, we propose to obtain cluster-centers in such way that the following three conditions are satisfied.

```
Insert($G, covered(1:n), v$):
  $G$ stores the similarity graph with adj data;
  $covered$ stores coverage count of each object;
  $v$ is the id of the new object to be inserted

1.  add $v$ to $G$, update $covered(v)$
2.  if $covered(v) > 0$ /* already covered */
3.     MakeCenterIfHasIllegalNeighbors($v$)
4.     if $(is\_center(v)) == false$)
5.        MakeCenterIfHasHigherDegree($v$)
6.  else /*v is NOT covered */
7.     forall $x \in adj(v)$
8.        MakeCenterIfHasIllegalNeighbors($x$);
9.     if $covered(v) == 0$ /*still not covered */
10.       HighestDegreeGetsCenter($v$)

MakeCenterIfHasIllegalNeighbors($v$)
1.  forall $x \in adj(v)$
2.    MakeNonCenterIfIllegal($x$)
3.  if (center-removed == true )
4.    MakeCenterAndUpdate($v$)
```

Fig. 4 Dynamic insert algorithm

1. Every object is either a cluster-center or is adjacent to at least one cluster center.
2. A cluster center with $degree > 0$ must be adjacent to at least one non-center object of smaller or equal degree.
3. No cluster-center is redundant, i.e., every cluster-center covers at least one object exclusively.

Note that the first of the above conditions is from the definition of LBSC. However, the second and the third conditions are chosen for LBSC to have a reasonably good solution in a dynamic setting. In the experimental section, we shall show that the above dynamic algorithm generates less centers in comparison with both static and dynamic star clustering algorithm.

4.1 Inserting a new object

We first define some terms before describing our insert algorithm. A center is called *illegal* if it does not satisfy condition (2) above, i.e., all its adjacent vertices have degree strictly higher than it. A center is called *redundant*, if its removal does not change (increase) the number of nodes that need to be covered.

Figure 4 shows the pseudo-code to insert a new vertex v. It also accepts the similarity graph and the *covered* vector as parameters; the latter stores the current coverage of each existing vertex. Once a new object v is added, the adjacency list of similarity graph G is updated. If v is adjacent to any existing center, it is properly reflected in the *covered* vector (line 1). Now, we have two different cases to consider.

In the first case, when v is covered, condition 1 is already satisfied. Then, we check condition 2 (illegal center) for all the centers that are adjacent to v. Note that, since the addition of new vertices changes the adjacency list of some of the existing vertices, this may change one legal center to illegal. If this check succeeds (some illegal neighboring center is found), we make the illegal center as non-center and v becomes a center. The above step is performed by the subroutine on line 3 and it is called recursively on success (in line 5 of Fig. 5). In

```
MakeCenterAndUpdate($v$):
1.  forall $x \in adj(v)$
2.     if ($is\_center(x)$)
3.        RemoveCenterIfRedundant($x$)
4.     else
5.        MakeCenterIfHasIllegalNeighbors($x$)
```

Fig. 5 Make center routine

```
Delete($G, covered(1:n), v$):
   $G$ stores the similarity graph with adj data
   $covered$ stores coverage count of each object
   $v$ is the id of the existing object to be deleted

1.   delete $v$ to $G$, update $covered(v)$
2.   if $was\_center(v)$ /* was a center before deletion */
8.      forall $x \in adj(v)$
4.         if ($degree(x)$) == 0)
5.            MakeCenter($x$)
6.         else if covered($x$) == 0
7.            HighestDegreeGetsCenter($x$);
8.   else /* v was not a center before deletion*/
9.      forall $x \in adj(v)$
10.        if $is\_center(x)$
11.           RemoveCenterIfRedundant($x$)
12.        else
13.           forall $y \in adj(v)$
14.              MakeCenterIfHasIllegalNeighbors($y$)
```

Fig. 6 Dynamic delete algorithm

case the above step fails (v is not a center yet), we test whether v should be a center as it has strictly higher degree than any of its neighboring centers (using subroutine on line 5). Note that this step is not essential for correctness according to our conditions, but this heuristic has potential to generate better centers. For the second case, when v is not covered, then none of its adjacent vertices is a center and to fulfil the coverage requirements, at least one of these vertices should be a center. We first test whether it should be some $x \in adj(v)$ by checking whether it has any illegal neighbors. If not, then we choose the vertex with the highest degree as a center (using subroutine on line 10). The body of one of the subroutines is also shown, and the other two subroutines are very straightforward (not shown).

When a node is made a center, some auxiliary updates are required which is shown in Fig. 5. First, redundancy check is required for all other centers that are adjacent to it to satisfy the condition 3. Moreover, some adjacent non-centers can also become a center as one of its neighboring center becomes illegal. Figure 7 shows an example of insertion, where we explain the steps of the insertion algorithm.

4.2 Deleting an existing object

Deletion is comparably easier. Like insertion, we first update the adjacency and covered vector as necessary. Then, we consider the following two cases. The first is, when the deleted node v was a center. In that case, we first need to check if any of its adjacent vertex becomes

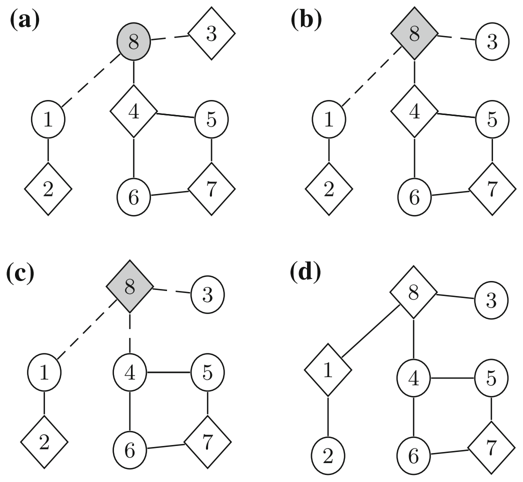

Fig. 7 Insertion example; a *diamond shape* indicates a center. **a** After inserting node 8, node 3 is illegal center. **b** Node 3 becomes non-center and node 8 becomes a center; now, node 4, a center-neighbor of 8 is redundant. **c** Redundant center removed; a non-center neighbor (of 8), 1, has an illegal center 2 as neighbor. **d** Final result; an offline algorithm will produce identical result in this case

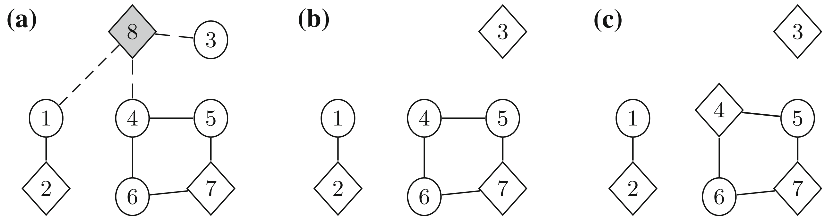

Fig. 8 Deletion example; a *diamond shape* indicates a center. **a** Node 8, a center node, is to be deleted. After deletion, the node 3, an isolated node, will be a center by default; node 4 is yet to be covered. **b** Node 8 is deleted, node 3 became a center. To cover node 4, one more center is required to be elected. The highest degree node be comes a center rule would be followed. **c** Node 4 is a center to cover itself; node 4 and all its adjacent nodes have the same degree value, so node 5 or node 6 could be a center as well instead of the node 4

isolated. All of those become a center. For the remaining, if they are still covered, we return immediately. Assume x is not covered, then we cover x by making the highest degree vertex (among itself and its neighbors) a center. Figure 8 shows a deletion example to explain these steps. For the second case, when v was not a center, its removal does not violate condition (1), so we check for condition (2) and (3) by calling the same methods as in insert routine. The pseudo-code is provided in Fig. 6.

4.2.1 Complexity analysis

Complexity of dynamic *SimClus* is difficult to analyze, since it depends on the degree of the vertices that are adjacent to the inserted (or deleted) vertex. For the sake of average-case analysis, we assume Erdos-Renyi random graph model, where the (expected) degree of a node is equal to $d = p.|V|$; p is the probability of an edge to exist. The following analysis is made based on this d value.

The redundancy checking of a node takes $O(d)$ time as it just reads the *covered* array of its adjacent nodes. Similarly, the illegal center condition checking of one node also takes $O(d)$ time, as it reads the degree of the adjacent nodes. So, considering the above check for all the nodes adjacent to the inserted (deleted) node, the total cost is $O(d^2)$. But, the **MakeCenterIfHasIllegalNeighbors** subroutine is called recursively on success. If the recursion is successful for I times, the total cost is $O(Id^2)$. But, generally the value of I is very close to 1 as we shall prove below.

By definition, an illegal center has the smallest degree than all its neighbors. Assume u is a node, $deg(u)$ is its degree and $adj(u)$ is its adjacency list. The probability that u is an illegal node is:

$$= \sum_{i=1}^{n-2} P\left[\forall v \in adj(u) : deg(v) \geq i+1 \mid deg(u) = i\right]$$

$$= \sum_{i=1}^{n-2} \left(P\left[deg(v) \geq i+1 | v \in adj(u)\right]\right)^i \cdot P\left[deg(u) = i\right]$$

$$= \sum_{i=1}^{n-2} \left(\sum_{k=i}^{n-2} \binom{n-2}{k} p^k (1-p)^{n-k-2} \right)^i \cdot \binom{n-1}{i} p^i (1-p)^{n-i-1}$$

The probability that the node u has degree i is equal to $\binom{n-1}{i} p^i (1-p)^{n-i-1}$ and the probability that a node $v \in adj(u)$ has degree greater than i is equal to $\sum_{k=i}^{n-2} \binom{n-2}{k} p^k (1-p)^{n-k-2}$. For a node to be illegal, all its neighbors should have degree greater than its degree. Thus, if u has degree i, its probability to be illegal is equal to the argument of the outer sum of the last line of the above set of equations. Since an illegal node can have degree from 1 to $n-2$, the outer sum aggregates these probabilities. If the above probability is p_{repeat}, using geometric distribution, the average value of I is $\frac{1}{1-p_{repeat}}$. For all possible values of p, the value for I is less than 2 (found with direct calculation in Matlab) when $n \geq 10$. For example, the highest value of I from $n = 10$ is around 1.75 and for $n = 1000$, it is around 1.015 and with larger n, the value diminishes further.

Intuitively, it means that for random graphs, every node is generally connected to some vertices with higher and some vertices with lower degrees than its degree; hence, very few nodes are illegal. For real-life graphs, the constant may be a bit higher. So, considering I a constant, the complexity of dynamic algorithm for one insertion or deletion is $O(d^2)$.

5 Experiments

The objective of the experiments is to show the following. First, the **SimClus** clustering returns a smaller number of representatives for LBSC, yet it yields similar quality clustering. Second, the representative objects are of good quality and can generate clustering that is better than the traditional clustering algorithms. Finally, that the LBSC approach is particularly suitable for clustering multi-label data sets. We also include some results from a real-life e-commerce data set.

Besides **SimClus** and *star*, the experiments use two other traditional clustering algorithms: k-Medoids, and UPGMA (Unweighted Pair Group Method with Arithmetic Mean) which is a hierarchical clustering approach. For all the algorithms, the document similarity is modeled by using cosine similarity and a similarity matrix is provided as input. For k-Medoids, we use our own implementation which works very similar to k-Means, but in each iteration, when k-Means chooses a new center by averaging the vectors in that cluster, k-Medoid chooses it by finding the object with the best average similarity to all the objects in that cluster (thus, it has a quadratic time complexity). For the UPGMA hierarchical clustering algorithm, we use

Table 1 Performance on random graphs

Vertex	Edge	Number of clusters			
		SimClus		Star	
		Static	Dynamic	Static	Dynamic
Erdos-Renyi random graphs					
1000	5000	144	194	209	209
1000	10000	84	123	126	130
1000	100000	16	22	21	24
10000	50000	1424	1951	2066	2068
10000	100000	840	1227	1323	1311
10000	1000000	147	225	247	245
Power law graphs					
1000	5000	547	564	579	579
1000	10000	525	539	550	549
1000	100000	496	502	502	503
10000	50000	3285	3498	4015	4024
10000	100000	2734	2928	3340	3340
10000	1000000	1991	2094	2156	2156

the implementation available in the Cluto Package.[2] The star algorithm was implemented by following the pseudo-code provided in the paper [2].

5.1 Synthetic data: random graphs

The first experiment considers synthetic data, in the form of random similarity graph of various sizes and types for both static and dynamic scenario. For the dynamic experiments, we shuffle the vertices of the similarity graphs randomly and insert them in the existing graph (staring from an empty graph) in that order. With a small probability (.05), we also delete a random vertex after every insertion. For random graph type, we consider the following two model: (1) Erdos-Renyi model and (2) Power-law graph model.

The result is shown in Table 1 for these two models. For both static and dynamic versions, **SimClus** achieves better optimization (smaller number of clusters) in comparison with Star clustering algorithm. Specifically, the static version significantly outperforms the static (and dynamic) version of star clustering. As expected, our dynamic version cannot perform as good as our static version, yet it performs better than the star clustering algorithm.

5.2 Newsgroup data set

For real-life data set, we chose the Twenty-newsgroup data set from the UCI Machine Learning Repository (http://www.ics.uci.edu/~mlearn). This data set is interesting for our experiments, as it has a set of documents that have multiple labels. We used the rainbow package[3] to convert the documents into word vectors and then used the best 100 words (in terms of

[2] http://glaros.dtc.umn.edu/gkhome/views/cluto/index.html.

[3] http://www.cs.cmu.edu/~mccallum/bow.

mutual information gain statistics) as feature vectors of those documents. We discarded any document that did not have any of the top 100 words. The final data set had 16701 documents. Out of them, 16199 had a unique label. 497 documents had 2 labels; only 5 documents had more than 2 labels; and, as the name suggests there are 20 different labels, in total.

In this experiment, we cluster the documents using different clustering algorithms and compare the clustering performance using supervised performance metrics, like precision, recall and F-measure. For k-Medoids and Hierarchical (UPGMA), we set the k value to be 20 to obtain 20 different clusters; then for every cluster, we use majority votes to label the cluster. For any object with k labels, $1/k$ vote is counted for all its labels. In case of **SimClus** and *star* algorithm, we cannot use k, so we cluster the documents for different similarity thresholds. Number of clusters are generally higher than 20 (exact values are shown in the k column). As the similarity threshold increases, the number of clusters also increases. Then we classify each of these clusters with the label of the cluster representative. Note that, many representatives (thus many clusters) can have same class-label; all of those clusters are collectively classified with that label. If the representative object has multiple labels, all the objects adjacent to that representative object get multiple labels. An object can also get multiple labels by being adjacent to multiple representatives with different class-labels.

We use both macro and micro measures for computing the supervised performance metric values like precision and recall. Macro measures are pivoted on the classes. So, a separate contingency table (actual vs predicted) is computed for each class. Then, the precision and recall value of each class are averaged to find the final metric values. On the other hand, micro measures are povited on the objects, so an aggregated contingency table is created from which the metrics are computed. While computing the values of different cells of the contingency tables, we treat each object uniformly. Thus, for an object with k different actual (or predicted) labels, its contribution to the appropriate cell is recorded as $1/k$ so that the aggregated contribution for different labels sums to 1.

Table 2 shows the results. Compared to k-Medoids and Hierarchical, both *star* and **SimClus** achieve much better performance in the F-score measure. It is mainly because of the very high recall that the latter algorithms achieve. The possible reason is that, instead of choosing exactly 20 centers (which may not be enough for the data set, since the cluster boundary is not regular), our algorithm samples enough representatives to cover the entire cluster. So, a document has much higher chance to be a neighbor of one of the many centers that matches with its class label. For example, if a document has a label "comp.graphics", out of, say 500 representatives, roughly $500/20 = 25$ representatives may have a label "comp.graphics". So, the document has a much higher probability to be a neighbor of any of these. One may expect a similar improvement in the performance of a k-Medoid or a hierarchical algorithm by applying a larger k value than the known number of classes (which is 20 for this data set). But, there exist two challenges. First, it requires to design a cluster-merging algorithm to obtain the desired number of clusters (20 for this data set) at the end. Second, it may be difficult to choose a suitable value of k.

In comparison between *star* and **SimClus**, their F-values are similar. However, **SimClus** chooses 30–40% less centers compared to the star algorithm. We mentioned earlier that the number of the representatives is significant in representative-based classification, as the labels of the representatives need to be manually evaluated, which is always costly.

Also note that for both star and SimClus, as the β value increases, the classification performance increases. It is because with a higher β value, there are more centers and more information is available to the algorithms as the labels of those centers. This also shows an application of LBSC clustering model as an active classification algorithm, where the labels of the objects are sought as needed to perform the classification through clustering. If more

Table 2 Comparison on newsgroup dataset

Algorithm	Parameters		Micro			Macro		
	β	k	Precision	Recall	F-Measure	Precision	Recall	F-Measure
k-Medoids	–	20	59.8	40.2	48.1	59.8	40.2	48.1
UPGMA	–	20	48.4	37.3	42.1	48.4	37.3	42.1
Star	0.50	125	36.5	94.5	52.7	39.1	94.4	55.3
	0.55	183	39.5	94.5	55.7	42.1	94.3	58.2
	0.60	273	45.0	95.0	61.0	45.7	94.8	61.7
	0.65	432	51.9	95.9	67.3	52.6	95.7	67.9
	0.70	668	59.8	96.6	73.9	60.5	96.5	74.3
	0.75	1060	67.6	96.8	79.6	67.9	96.7	79.8
	0.80	1811	77.1	97.3	86.1	77.1	97.2	86.0
SimClus	0.50	75	37.6	93.3	53.6	40.8	92.9	56.7
	0.55	106	40.7	92.8	56.6	42.4	92.6	58.1
	0.60	147	45.0	92.8	60.6	48.0	92.6	63.3
	0.65	229	50.7	92.4	65.5	52.3	92.1	66.7
	0.70	391	54.7	93.1	68.9	55.9	92.9	69.8
	0.75	699	64.0	95.0	76.4	64.9	94.8	77.1
	0.80	1274	70.5	96.1	81.4	70.5	96.0	81.3

information is available, better classification can be achieved. In case of clustering, choosing higher value does not yield better clustering as with higher value it can happen that a desired cluster can be fragmented. So, if a size (number of clusters) insensitive performance metric is available for the clustering, one should try to find the optimal value of β that would yield the best clustering. If the metric function is convex, a binary search can be performed to find the global optimal value of $beta$. Also one can choose to use comparably large $beta$ value to find many clusters and then merge them based on pairwise similarity of the original clusters to obtain the final clustering. This is a very effective approach as shown in [10,16,25].

Our third set of experiments compares the ability of **SimClus** and *star* to predict multiple labels on the newsgroup data set. For this, we find the predicted labels of the objects that has more than one actual labels (502 documents qualify). An actual multi-label is considered as recalled, if at least two predicted labels match with two of the actual labels. We name the recall measure that is computed in this way as *multi-recall*. Table 3 shows the result for three different values of similarity threshold. In this table, we compare the number of centers and the multi-recall values between *SimClus* and *star* algorithm. The recall values for both the algorithms drop with increasing threshold values. The reason behind that is with large threshold, β-similarity graphs become more and more sparse, so an object is not connected to many representatives. Thus, its ability to predict multiple levels diminishes. In comparison with *star*, **SimClus** performs substantially better than *star* for a β value of 0.60 even with 47% less centers. As β increases, it loses to *star*, mainly because it optimizes the number of centers. In fact, as we investigated we found that *star* achieved better multi-recall, by actually selecting many of the multi-labeled nodes as the center objects.

We also compare the timing performance of our algorithm in comparison with other algorithms. The result is shown in Table 4. The table only shows the execution time (in seconds) of the clustering task; the I/O cost of loading the similarity matrix is excluded for the sake of

Table 3 Overlapping clustering performance

Similarity threshold (β)	SimClus		Star	
	Centers count	Multi-recall	Centers	Multi-recall
0.60	144	76.49	273	48.41
0.70	382	41.63	550	45.80
0.80	1274	28.88	1811	31.47

Table 4 Execution time comparison (time shown in seconds)

Algorithm	k or β	Size			
		4000	8000	12000	16701
k-Medoids	20	33.36	301.21	1200.12	3100.38
UPGMA	20	6.71	33.48	75.76	216.96
RB	20	25.64	116.68	297.88	677.28
Star (*static*)	0.50	0.02	0.04	0.06	0.09
	0.75	0.02	0.04	0.05	0.08
SimClus (*static*)	0.50	5.09	27.62	68.68	197.49
	0.75	0.50	2.53	6.78	15.22

fair comparison. The time is reported for a 2.1 GHz machine with 2GB RAM with Linux OS. For this experiment, we randomly select documents from the newsgroup data set to make 3 different smaller-size data sets (4000, 8000, 12000). Besides k-Medoids, and UPGMA, we also use another algorithm, RB (from the Cluto software), which is a fast partitional algorithm that uses repeated bisection. For k-Medoids, UPGMA and RB, which require a k value to cluster, $k = 20$ is used as it is the natural clustering for the newsgroup data set. For *star* and **SimClus**, timing for two different similarity thresholds (0.5 and 0.75) are reported. Different threshold accounts for different number of edges in the β-similarity graphs.

From the Table, we see that *star* is the fastest among all the algorithms as it just needs to sort the vertex-set only once based on their degree and its timing varies negligibly based on the number of edges. **SimClus** came out to be the second best and its timing varies based on the number of edges in the similarity graph since the time complexity has an $|E|$ term. The execution time of **SimClus** for 0.5 threshold (which is very relaxed for LBSC) with the entire newsgroup data set is 1.1, 3.4, and 15.7 times better than UPGMA, RB, and k-Medoids, respectively. We also compared the execution time of our dynamic algorithm to evaluate its utility over the static algorithm. For the newsgroup data set, average insertion time of one document is 0.13 second, which is much faster in comparison with the re-clustering time (15.22 seconds) using the static algorithm.

5.3 eCommerce data set

For the final evaluation of LBSC clustering, we use a subset of the eCommerce data set that is used in [26]. It includes approximately 7.9 million customer queries for product search in eBay. To represent the similarity among the queries, a similarity graph is computed by considering the textual similarity and user-session similarity. The first is just a similarity metric based on the number of common words between a pair of queries, and the second is based on the frequency with which the two queries co-occurred in a single user session from historical eBay data set. Using efficient word indexing, the similarity graph is computed in almost linear time. For details on how the similarity is computed, see the original paper [26]. The graph is very sparse; it contains approximately 20.6 millions edges. The edges are weighted, with the weight values between (0, 1], average weight value is 0.13. The graph shows power-law degree distribution.

Our objective in this research is to cluster the queries to find representative set of queries. Clustering is in general useful to categorize the queries which can further be utilized to obtain a catalog. But, for eBay, finding representative objects of the clusters has some other significant use cases, such as (1) related search recommendations, (2) finding fall-back queries if the original query has null or very small result-set.

Since the data set is already in a graph format, the LBSC formulation is particularly suitable for this task. We run **SimClus** with $\beta = 0.1$, which yields about 650K representative objects. The clustering task finished in approximately 3 days. As an alternative, we also tried to cluster the data set with MCL graph clustering algorithm [12]. It took more than a week on a hadoop-based cluster of 8 identical machines. It returned nearly a million clusters with its default settings.

Since the number of clusters are different, it is difficult to compare between the quality of the clusterings returned by **SimClus** and MCL by different graph-clustering metrics, like average normalized cut, modularity, etc. But with a visual inspection, it seems that the results are comparable. Most of the clusters are in the form of stars with small sizes except few very large (and dense) clusters of size about 10 thousand. Such a result is expected as the query graph has scale-free nature. So, there is a small set of very popular queries (like ipod, antiques, nike) that have numerous similar queries that built the large clusters. On the other hand, majority of the queries do not have many neighbors, thus forming a large number of small clusters. However, **SimClus** is better than MCL, since it provides the representatives instantly. Furthermore, it allows overlapping clustering which is very useful in this clustering problem. For instance, a query like **ipod battery nano** is clustered in both **nano ipod** and in **ipod battery** cluster using LBSC. But, MCL places it only in one cluster. A small snippet of these clusters obtained from **SimClus** is shown in Fig. 9.

6 Related work

Lower bound similarity clustering did not get much attention in the past. *star* clustering [2] is the leading algorithm in this paradigm. [14] highlighted some of the problems of *star* clustering algorithm and proposed an alternative solution to improve the *star* clustering; however, they did not provide any approximation result that their algorithm achieves. Furthermore, they considered only the static scenario. LBSC is naturally aligned with overlapping clusters. For example, in a work regarding gene functional classification [16], the authors used a bound on the kappa statistic as a similarity threshold to find all possible overlapping subset of genes as initial clusters. From this initial clustering, any two clusters that have more than 50% common objects were merged recursively. Authors showed that such an approach generates much superior (biologically meaningful) clusters in comparison with k-Means, or hierarchical clustering. Note that in this clustering approach, the lower bound is applied between every pair of objects. So, in a graphical representation, the initial clusters are just cliques. In a seminal work on community finding [25], the authors used k-clique ($k = 6$) to obtain initial overlapping clusters in a social network. Then, any two adjacent k-cliques (adjacent k-clique's share $k - 1$ vertices) were merged to obtain the final clustering. Both the above clustering approaches are examples of finding overlapping clusters from similarity graph that

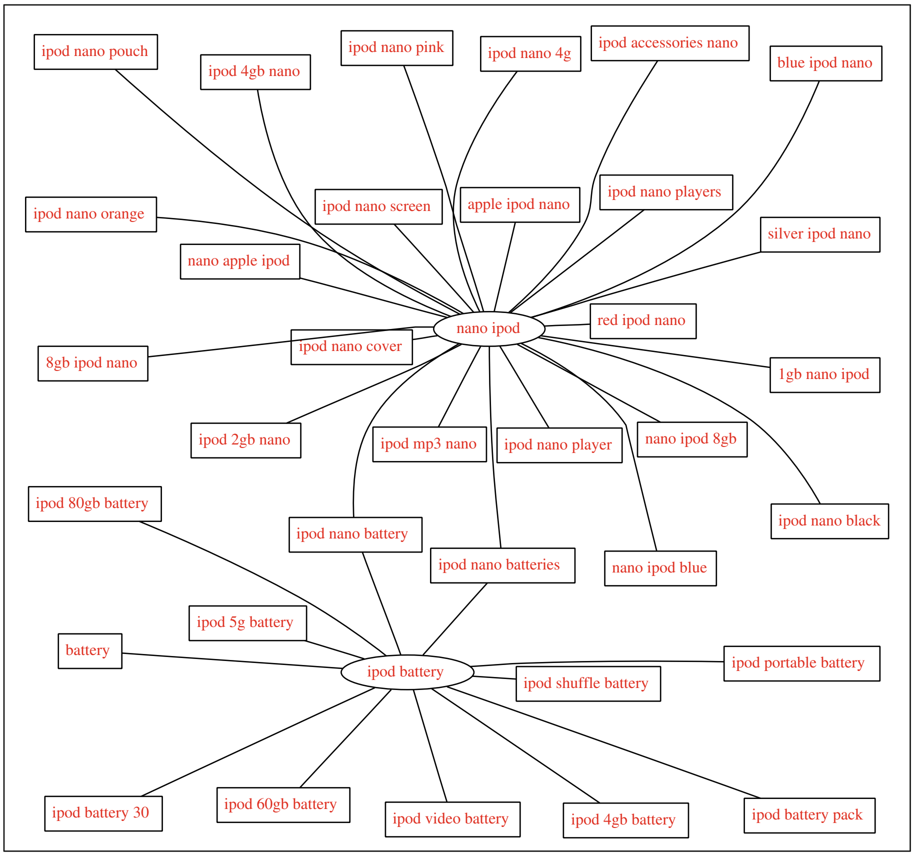

Fig. 9 Part of two star clusters obtained using SimClus algorithm are shown; the centers are oval shaped, only the edges between the center and the satellite nodes are shown

was obtained by lower bounding some form of similarity measure. The initial clusters are then merged to obtain the final clustering. Merging generates fewer clusters, but it loses the representative objects. So, our approach is suitable if a representative object is desired.

Fuzzy clustering is one of the earliest approach that explicitly formulated the concept of overlapping clustering. In this domain, Fuzzy c-Means [11] is the flagship clustering algorithm. It is equivalent to soft k-Means algorithm obtained by applying EM to a mixture of Gaussian models. Objects are assigned to a cluster in a similar way as in soft k-Means by applying a threshold to the posterior probability obtained through soft k-Means. Recently, probabilistic relation models were also applied to obtain overlapping clusters [7]. However, model-based algorithms can fail if the data does not agree to the model's assumption.

Obtaining representative objects through clustering is well studied in signal processing, and also in theoretical computer science. In fact, the original k-Means clustering algorithm was proposed by signal-processing community for vector quantization [21], which is used for lossy data compression. The k-Medoids clustering is studied extensively in theoretical computer science and optimization [6]. Most of the formulations work for a given k, and their objective is to minimize the maximum distance from an object to its nearest center.

In terms of dynamic clustering algorithms, [3] finds the online version of the popular EM-based approach. [1] solve it by using an online component and an off-line component, [4] proposed an online variant of a different mixture model clustering for text data. One may consider streaming model as dynamic clustering; however, this model imposes very strict requirements [5,20] on available memory and the number of passes over the data. Hence, the approximation quality is generally poor.

7 Conclusions

In this paper, we proposed a clustering algorithm that uses lower bound on similarity to cluster a set of objects from the similarity matrix. Experiments on real-life and synthetic data sets show that the algorithm is faster and produces higher quality clustering in comparison with existing popular algorithms. Furthermore, it provides representative centers for every cluster; hence, it is effective in summarization or semi-supervised classification. It is also suitable for multi-label or on-line clustering.

Acknowledgments We like to thank Nish Parikh of eBay Research Labs for the eCommerce data set. Saeed Salem was supported by NIH Grant Number P20 RR016471 from the INBRE Program of the National Center for Research Resources.

References

1. Aggarwal CC, Han J, Wang J, Yu PS (2003) A framework for clustering evolving data streams. In: VLDB proceedings
2. Aslam J, Pelekhov JE, Rus D (2004) The star clustering algorithm for static and dynamic information organization. Graph Algorithms Appl 8(1):95–129
3. Azoury KS, Warmuth MK (2001) Relative loss bounds for on-line density estimation with the exponential family of distributions. Mach Learn 43(3):211–246
4. Banerjee A, Basu S (2007) Topic models over text streams: A study of batch and online unsepervised learning. In: Proceedings of SIAM data mining conference 2007
5. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data streams. In: SIGMOD-SIGACT-SIGART symposium on principles of database systems, ACM 2002
6. Badoiu M, Har-Peled S, Indyk P (2002) Approximate clustering via core-sets. Proc. of thirty-fourth STOC, ACM, pp 250–257
7. Banerjee A, Krumpelman C, Basu S, Mooney RJ, Ghosh J (2005) Model-based overlapping clustering. In: Eleventh ACM SIGKDD international conference on KDD
8. Boyardo RJ, Ma Y, Srikant R (2007) Scaling up all pairs similarity search. Proc. of 16th Internation World Wide Web Conference, May 2007
9. Bringmann B, Zimmermann A (2009) One in a million: picking the right patterns. Knowl Inf Syst 18(1):61–81
10. Chaoji V, Hasan MA, Salem S, Zaki MJ (2009) SPARCL: an effective and efficient algorithm for mining arbitrary shape-based clusters. Knowl Inf Syst 21(2):201–229
11. Cox E (2005) Fuzzy modeling tools for data mining and knowledge discovery. Morgan Kaufmann
12. Dongen SV (2000) Graph clustering by flow simulation. PhD in Computer Science, Univeristy of Utrecht
13. Chen X, Zhang D, Lee WS (2005) Text classification with kernels on the multinomial manifold. SIGIR Proceedings
14. G-Garcia R, Badia-Contelles J, Pons-Porrata A (2003) Extended star clustering, in CIARP (LNCS 2905). Springer, New York
15. Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of np-completeness. W.H. Freeman, San Francisco
16. Huang D, Sherman B, Tan Q et al (2007) The DAVID gene functional classification tool: a novel biological module-centric algorithm to functionally analyze large gene lists. Genome Biol 8(9)
17. Kobayashi M, Aono M (2006) Exploring overlapping clusters using dynamic re-scaling and sampling. Knowl Inf Syst 10(3):295–313

18. Karypis G, Han E, Kumar V (1999) Chameleon: hierarchical clustering using dynamic modeling. Computer 32(8):68–75
19. King G, Tzeng W (1997) On-line algorithm for the dominating set problem. Inf Process Lett 61:11–14
20. Liu H, Lin Y, Han J (2009) Methods for mining frequent items in data streams: an overview. Knowl Inf Syst. doi:10.1007/s10115-009-0267-2
21. Lloyd SP (1982) Least square quantization in pcm. IEEE Trans Inf Theory 28(2):129–136
22. Lund C, Yannakakis M (1994) On the hardness of approximating minimization problems. J ACM 41(5):960–981
23. Narasimhamurthy A, Greene D, Hurley N, Cunningham P (2009) Partitioning large networks without breaking communities. Knowl Inf Syst. doi:10.1007/s10115-009-0251-x
24. Ng RT, Han J (1994) Efficient and effective clustering methods for spatial data mining. In: VLDB proceedings August 1994
25. Palla G, Derenyi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435
26. Parikh N, Sundaresan N (2008) Inferring semantic query relation from collective user behavior. In: Proceedings of conference on information and knowledge management, ACM, October 2008
27. Vazirani VV (1998) Approximation algorithms. Springer, New York
28. Xin D, Han J, Yan X, Cheng H (2005) Mining compressed frequent-pattern sets. In: VLDB proceedings, August 2005
29. Zamir O, Etzioni O (1994) Grouper: a dynamic clustering interface to web search results. In: Proceedings of world wide web, pp 1261–1274
30. Zuckerman D (1993) Np-complete problems have a version that's hard to approximate. In: Proceedings of eighth annual structure in complexity theory, IEEE Computer Society, pp 305–312

Author Biographies

Mohammad Al Hasan is an Assistant Professor of Computer Science at Indiana University–Purdue University, Indianapolis (IUPUI). Before that, he was a Senior Research Scientist at eBay Research Labs, San Jose, CA. He received his Ph.D. degree is Computer Science from Rensselaer Polytechnic Institute (RPI) in 2009. His research interest focuses on developing novel algorithms in data mining, data management, information retrieval, machine learning, social network analysis, and bioinformatics.

Saeed Salem has been an assistant professor of Computer Science at North Dakota State University since August 2009. He received his Ph.D. in Computer Science from Rensselaer Polytechnic Institute, New York in 2009. His research interests are in data mining, and bioinformatics.

Mohammed J. Zaki is a Professor of Computer Science at RPI. He received his Ph.D. degree in computer science from the University of Rochester in 1998. His research interests focus on developing novel data mining techniques, especially in bioinformatics. He has published over 175 papers and book-chapters on data mining and bioinformatics. He is currently an Executive Editor for Statistical Analysis and Data Mining, and an Associate Editor for Data Mining and Knowledge Discovery, ACM Transactions on Knowledge Discovery from Data, Knowledge and Information Systems, ACM Transactions on Intelligent Systems and Technology, and International Journal of Knowledge Discovery in Bioinformatics. He was the program co-chair for SDM'08, SIGKDD'09 and PAKDD'10. He received the NSF CAREER Award in 2001 and the DOE Career Principal Investigator Award in 2002. He also received the ACM Recognition of Service Award in 2003 & 2009, and an IEEE Certificate of Appreciation in 2005.